MethodSCRIPT SDK Example - Arduino

PalmSens
Compact Electrochemical Interfaces

Last revision: April 01, 2019

www.palmsens.com

1.1 Contents:

The arduino example *MethodSCRIPTExample.ino* found in the */MethodSCRIPTExample_Arduino/MethodSCRIPTExample* folder demonstrates basic communication with the EmStat Pico through Arduino MKR ZERO using the MethodSCRIPT SDK (C libraries). The example allows the user to start measurements on the EmStat Pico from a Windows PC connected to the Arduino through USB.

1.2 Hardware setup:

- To run this example, connect your Arduino MKRZERO *Serial1* port Rx (pin 13), Tx (pin 14) and GND to the EmStat Pico *Serial* Tx, Rx and GND respectively.
- Make sure the UART switch block SW4 on the EmStat Pico dev board has the switches for MKR 3 and 4 turned on.
- The Arduino board should be connected normally to a PC.
- If not powering the EmStat Pico by other means, it should be connected to the PC through USB for power.

1.3 Environment setup:

- To run this example, you must include the MethodSCRIPT C libraries first.
- To do this, follow the menu *Sketch -> Include Library -> Add .ZIP/Library* and select the *MethodSCRIPTComm* folder.

1.4 How to use

- Compile and upload this sketch through the Arduino IDE.
- Next, open a serial monitor to the Arduino (you can do this from the Arduino IDE).
- You should see messages being printed containing measured data values from the EmStat Pico.

1.5 Communications

The MSComm.c from the MethodSCRIPT SDK (C libraries) acts as the communication object to read/write from/to the EmStat Pico. In order to use the C library, *MSComm*, the *extern C* wrapper has to be used because Arduino uses a C++ compiler.

```
extern "C" {
  #include <MSComm.h>
  #include <MathHelpers.C>
};
```

As MSComm is the communication object with the EmStat Pico it needs some read/write functions to be passed in through the *MSCommInit()*. However, because the C compiler doesn't understand C++ classes, the write/read functions from the Serial class are wrapped in a normal function, first as shown below.

```
int write_wrapper(char c)
{
  if(_printSent == true)
  {
    //Send all data to PC as well for debugging purposes
    Serial.write(c);
  }
  return Serial1.write(c);
}

int read_wrapper()
{
```

```
  int c = Serial1.read();

  if(_printReceived == true && c != -1) //-1 means no data
  {
    //Send all received data to PC for debugging purposes
    Serial.write(c);
  }
  return c;
}
```

The *MSComm* library has to be initiated with these read/write functions as shown below.

```
MSComm _msComm;
RetCode code = MSCommInit(&_msComm, &write_wrapper, &read_wrapper);
```

A new UART instance has to be created and assigned to TX (14) and RX (13) pins on the Arduino.

```
Uart Serial1(&sercom5, 14, 13, SERCOM_RX_PAD_3, UART_TX_PAD_2);
```

1.5.1 Connecting to the device

The code within the setup() function is executed only once.

The Serial Data Line (SDA) and Serial Clock Line (SCL) are assigned to the pins 13 and 14 respectively.

```
//Assign SDA (serial data line) function to pin 13
pinPeripheral(13, PIO_SERCOM_ALT);
//Assign SCL (serial clock line) function to pin 14
pinPeripheral(14, PIO_SERCOM_ALT);
```

In order to begin communication, the serial ports are initiated with the baud rate 230400.

```
Serial.begin(230400);
Serial1.begin(230400);
```

'Serial1' is the port on the EmStat Pico dev board set to communicate with the Arduino which in turn communicates with the EmStat Pico and 'Serial' is the port on the Arduino communicating with the PC.

1.5.2 Sending the MethodSCRIPT

The MethodSCRIPT can be either stored in a sd card in the Arduino and read from it or stored in a constant string. In the example, the MethodSCRIPT is stored in a constant char array as shown below.

```
//LSV measurement configuration parameters
const char* LSV_METHOD_SCRIPT = "e\n"
                                "var c\n"
                                "var p\n"
                                "set_pgstat_mode 3\n"
                                "set_max_bandwidth 200\n"
                                "set_cr 500u\n"
                                "set_e -500m\n"
                                "cell_on\n"
                                "wait 1\n"
                                "meas_loop_lsv p c -500m 500m 50m 100m\n"
                                "pck_start\n"
                                "pck_add p\n"
                                "pck_add c\n"
                                "pck_end\n"
                                "endloop\n"
                                "celloff\n\n";
```

The MethodSCRIPT is sent to the Arduino and in turn to the EmStat Pico through the *Serial1* port. The example uses the *MSComm* library to perform the write operation. This function requires a reference to the initiated *MSComm* struct (_msComm) to be passed along.

```
void SendScriptToDevice(const char* scriptText)
{
  WriteStr(&_msComm, scriptText);
}
```

1.5.3 Receiving the data packages

This example uses the *MSComm* library to receive and parse the data returned by a measurement. In order to read and parse the measurement data packages from the device, the *Receive Package()* from the *MSComm* library can be used. This function requires a reference to an initiated *MSComm* struct (*msComm*) and it returns the parsed data values in the referenced MeasureData struct (*data*)

```
code = ReceivePackage(&_msComm, &data);
```

1.5.4 Parsing the measurement data packages

Each measurement data package returned by the function *ReadBuf()* in *MSComm* library, can be parsed further to obtain the actual data values. Here's a set of data packages received from a Linear Sweep Voltammetry (LSV) measurement on a dummy cell with 10kOhm resistance.

```
eM0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing a measurement package, various identifiers are used to identify the type of package. For example, In the above sample,

1. 'e' is the confirmation of the "execute MethodSCRIPT" command.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a measurement data package.
4. "*\n" marks the end of a measurement loop.
5. "\n" marks the end of the MethodSCRIPT.

The data values to be received from a measurement can be sent through 'pck' commands in the MethodSCRIPT. Most techniques return the data values Potential (set cell potential in V) and Current (measured current in A). These can be sent with the MethodSCRIPT.

In case of Electrochemical Impedance Spectroscopy (EIS) measurements, the following *variable types* can be sent with the MethodSCRIPT and received as measurement data values.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following metadata values can also be obtained from the data packages, if present.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use at the moment)
- Noise (Noise)

1.5.4.1 Parsing the measurement data packages

Each measurement data package begins with the header 'P' and is terminated by a '\n'. The measurement data package can be split into data value packages based on the delimiter ';'. Each of these data value packages can then be parsed separately to get the actual data values.

The type of data in a data package is identified by its variable type:

- The potential readings are identified by the string *da*
- The current readings are identified by the string *ba*
- The frequency readings are identified by the string *dc*
- The real impedance readings are identified by the string *cc*
- The imaginary impedance readings are identified by the string *cd*

For example, in the sample package seen above, the *variable types* are
da7F85F3Fu - da Potential reading and
ba48D503Dp,10,288 – ba current reading.

The following 8 characters hold the data value. The data value for the current reading (8 characters) from the above sample package is *48D503Dp*.

The SI unit prefix from the package can be obtained from the parameter value at position 8

In the above sample package, the unit prefix for current data is 'p' which is 1e-12 A.

After obtaining variable type and the data values from the package, the metadata values can be parsed, if present.

1.5.4.2 Parsing the metadata values

The metadata values are separated based on the delimiter ',' and each of the values is further parsed to get the actual value.

The first character of each metadata value metaData[0] identifies the type of metadata.

'1' – status
'2' – Current range index
'4' - Noise

The metadata status is a 1 character hexadecimal bit mask.

For example, in the above sample, the available metadata values for current data are,
10,288. The first metadata value is 10.

1 – metadata status – 0 indicates OK.

The metadata type current range is 2 characters long hexadecimal value. If the first bit is high (0x80), it indicates a high-speed mode current range. The hexadecimal value can be converted to int to get the current range.

For example, in the above sample, the second metadata available is 288.

2 – indicates the type – current range
88 – indicates the hexadecimal value for current range index – 1mA. The first bit 8 implies that it is high speed mode current range.

Sample output:

Here's a sample measurement data package from a LSV measurement on a dummy cell with 10kOhm resistance and its corresponding output.

Pda7F85F3Fu;ba4BA99F0p,10,288

Output:
E (V) = -4.999E-01
i (A) = -4.999E-01
Status : OK
CR : 1mA (High speed)